Proceedings of Machine Learning Research vol 99:1–9, 2019 32nd Annual Conference on Learning Theory

Open Problem: The Oracle Complexity of Convex Optimization with Limited Memory

Blake Woodworth BLAKE@TTIC.EDU
Nathan Srebro NATI@TTIC.EDU
6045 S Kenwood Ave, Chicago, IL, 60637

Editors: Alina Beygelzimer and Daniel Hsu

Abstract

We note that known methods achieving the optimal oracle complexity for first order convex optimization require quadratic memory, and ask whether this is necessary, and more broadly seek to characterize the minimax number of first order queries required to optimize a convex Lipschitz function subject to a memory constraint.

1. Introduction

We consider first-order optimization methods for convex Lipschitz bounded functions. I.e., for the following optimization problem

$$\min_{x\in\mathbb{R}^d:\|x\|\leq B} F(x) \tag{1}$$

where F is convex and L-Lipschitz, we consider using (exact) queries returning $F(x)$ and a subgradient $\nabla F(x)$, and ask the classical question of how many queries are required to ensure we find an ϵ-suboptimal solution. The classical answer is that $O(d\log LB/\epsilon)$ suffice, using the center-of-mass method, and that, when $d \ll (LB/\epsilon)^2$ this is optimal (Nemirovsky and Yudin, 1983). However, the center-of-mass method is intractable, at least exactly. Other methods with a $O(\mathrm{poly}(d)\,\mathrm{polylog}(LB/\epsilon))$, and even $\tilde{O}(d\mathrm{log}(LB/\epsilon))$, query complexity and polynomial runtime have been suggested, including as the Ellipsoid method (Shor, 1970), Vaidya's method (Atkinson and Vaidya, 1995) and approximate center of mass using sampling (Bertsimas and Vempala, 2002). But these are generally not used in practice, since the higher order polynomial runtime dependence on the dimension is prohibitive. These methods also all require storing all returned gradients, or alternatively an ellipsoid in R^d, and so $\Omega(d^2)$ memory. A simpler alternative is gradient descent, which requires $O\left((LB/\epsilon)^2\right)$ queries, but only $O(d\log LB/\epsilon)$ memory and $O(d)$ runtime per query.

One might ask: is it possible to achieve the optimal query complexity using a "simple" method? Since it is much harder to provide runtime lower bound, we instead focus on the required memory, and ask: **is it possible to to achieve the optimal query complexity with $O(d\log LB/\epsilon)$ memory? How does the first-order oracle complexity trade off with the memory needed by an optimization algorithm?** This question is formalized in the following Section.

2. Problem Formulation

We capture the class of first-order optimization algorithms that use M bits of memory in terms of a set of "encoders" and "decoders." In each iteration, the decoder reads the M bits of memory, and determines a query point x_t. The encoder receives the function value $F(x_t)$ and a subgradient $\nabla F(x_t)$ at x_t—as is standard with oracle based optimization, if $F(x_t)$ is not differentiable at x_t, we require the method works for any valid subgradient used. The encoder then uses the current memory state, $F(x_t)$ and $\nabla F(x_t)$ to update the memory state for the next iteration. At the end, the algorithm's output is chosen as a function of the final memory state. To be clear, the encoding and decoding functions can require an arbitrary amount of memory to compute, and can compute using real numbers. However, between each access to the oracle, there is a "bottleneck" where the algorithm's state must be compressed down to M bits.

Formally, we define $\mathcal{A}_{M,T}$, the class of all deterministic[1] first-order optimization algorithms that use M bits of memory and T function value and gradient computations. An algorithm $A \in \mathcal{A}_{M,T}$ is specified by a set of decoder functions $\left\{\phi_t : \{0,1\}^M \to \mathbb{R}^d\right\}_{t=1}^{T}$, a set of encoder functions $\left\{\psi_t : \{0,1\}^M \times \mathbb{R} \times \mathbb{R}^d \to \{0,1\}^M\right\}_{t=1}^{T}$, and an output function $\zeta : \{0,1\}^M \to \mathbb{R}^d$. The algorithm's memory is initially blank $\mu_1 = 0$. The T iteration $t = 1...T$ are specified recursively by $x_t = \phi_t(\mu_t)$ and

$$\mu_{t+1} = \psi_t(\mu_t, F(x_t), \nabla F(x_t)) = \psi_t(\mu_t, F(\phi_t(\mu_t)), \nabla F(\phi_t(\mu_t))), \tag{2}$$

and the output of the algorithm, denoted $F(A)$, is given by:

$$A(F) = \zeta(\mu_T). \tag{3}$$

Let $\mathcal{F}_{L,B}^d$ be the set of all convex, L-Lipschitz functions $f : \mathbb{R}^d \to \mathbb{R}$ such that $\exists x^* \in \arg\min_x F(x)$ with $\|x^*\| \leq B$. We define the minimax memory-bounded first-order oracle complexity as

$$T_{L,B}(d, M, \epsilon) = \inf\left\{T \in \mathbb{N} \ : \ \inf_{A \in \mathcal{A}_{M,T}} \sup_{F \in \mathcal{F}_{L,B}^d} F(A(F)) - \min_{\|x\| \leq B} F(x) \leq \epsilon\right\}, \tag{4}$$

where the supremum over functions F should be interpreted also as a supremum over all valid subgradients $\nabla F(x_t)$ used in the updates. Without loss of generality, we will fix $L = B = 1$ and write $T(d, M, \epsilon) := T_{1,1}(d, M, \epsilon)$. We will further say that a query-memory tradeoff (T, M) is *possible* for a problem specified by (d, ϵ) if $T \geq T(d, M, \epsilon)$ and *impossible* if $T < T(d, M, \epsilon)$.

3. Current Knowledge

In high dimensions, when $d = \Omega(\frac{1}{\epsilon^2 \log 1/\epsilon})$, gradient descent is optimal in terms of both query and memory complexity, and so we consider only $d = O(1/\epsilon^2)$.

1. We focus on deterministic algorithms, but an analogous class of randomized algorithms could easily be specified. However, it seems unnecessary to complicate things in this way because there is evidence that there is little to be gained through randomization for solving problems of the form (1) Woodworth and Srebro (2017).

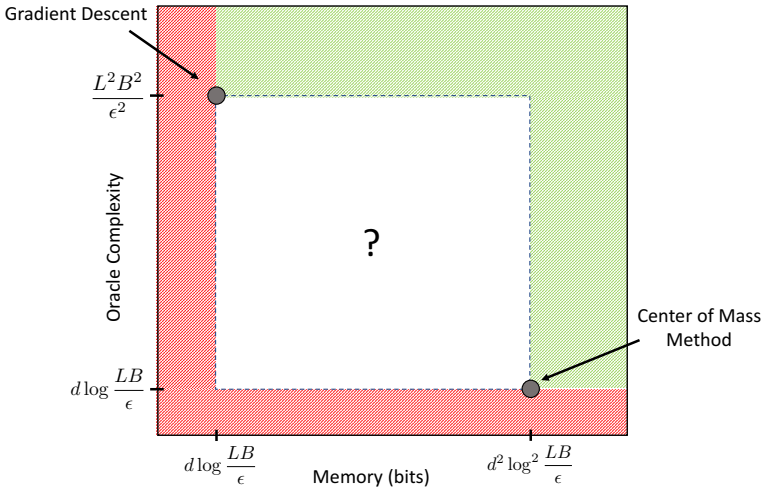

Figure 1: The tradeoff between the memory needed and the first-order oracle (query) complexity of optimization algorithms. The shaded red "L" shaped region along the bottom and left are trade-offs we know are impossible. The shaded green inverted-"L" shaped region along the top and right are trade-offs we know are possible. We do not know whether any trade-offs inside the "?" square are possible or not.

We can describe the minimax complexity $T(d, M, \epsilon)$, and the query-memory tradeoff, in terms of the regions of possible and impossible (T, M), as depicted in Figure 2. We currently understand only the extremes. With any amount of memory, $T = \Omega(d \log 1/\epsilon)$ queries are required, providing a lower bound for the possible region in terms of the query complexity (a horizontal lower bound in Figure 2). This is attained by the center of mass method, using $O(d^2 \log^2(1/\epsilon))$ bits of memory (see Appendix B for an analysis of Center of Mass with discrete memory), and so any $(T = \Omega(d \log 1/\epsilon), M = \Omega(d^2 \log^2 1/\epsilon))$ is possible (the rectangle above and to the right of "Center of Mass" in Figure 2). At the other extreme, even just representing the answer requires $\Omega(d \log(LB/\epsilon))$ bits (see Theorem 5 in Appendix C), providing a lower bound for the possible region in terms of memory (the vertical lower bound in Figure 2). This is attained by Gradient Descent using $O(1/\epsilon^2)$ queries (see Appendix A for an analysis of Gradient Descent with discrete memory), and so any $(T = \Omega(1/\epsilon^2), M = \Omega(d \log 1/\epsilon))$ is possible (the rectangle above and to the left of "Gradient Descent" in Figure 2).

To the best of our knowledge, what happens inside the square bordered by these regions is completely unknown. Nothing we know would contradict the existence of a $T, M = O(d \log 1/\epsilon)$ query and memory complexity algorithm, i.e. a single optimal method at the bottom left corner of the unknown square, making the entire square possible. It is also entirely possible, as far as we know, that improving over a query complexity of $\Theta(1/\epsilon^2)$ requires $\Omega(d^2 \log 1/\epsilon)$ memory, making the entire square impossible, and implying that no compromise is possible between the query requirement of Gradient Descent and memory requirement of Center of Mass.

4. Challenges

Ultimately, we would like fully understand what is and is not possible:

Question 1 ($500 or Two Star Michelin Meal) Provide a complete characterization of $T(d, M, \epsilon)$ and the possible (T, M) trade-off, preferably up to constant factors, and at most up to factors poly-logarithmic in T and M.

The most interesting scaling of d and ϵ is when the dimension is larger then poly-logarithmic but smaller then polynomial in $1/\epsilon$, so that $d \log 1/\epsilon$ memory is less then quadratic memory, but $1/\epsilon^2$ query complexity is not polynomial in d.

Even without understanding the entire trade-off, it would be interesting to study what can be done on its boundary. Perhaps the most important regime is the case of linear memory $M = \Theta\left(d \log \frac{LB}{\epsilon}\right)$. Therefore, as a starting point, we ask to characterize $T(d, \Theta\left(d \operatorname{polylog} \frac{LB}{\epsilon}\right), 1/\epsilon)$. In particular, is it possible to have query complexity polynomial in d with $\tilde{O}(d)$ memory?

Question 2 ($200 or One Star Michelin Meal) Can we have $T(d, M = \tilde{O}(d), \epsilon) = O(\operatorname{poly} d)$ when $d = \Omega(\log^c 1/\epsilon)$ but $d = O(1/\epsilon^c)$ for all c?

At the other extreme, we might ask whether quadratic memory is necessary in order to achieve optimal query complexity:

Question 3 ($200 or One Star Michelin Meal) Can we have $T(d, M = O(d^{2-\delta}), \epsilon) = \tilde{O}(d \operatorname{polylog} 1/\epsilon)$, for $\delta > 0$, when $d = \Omega(\log^c 1/\epsilon)$ but $d = O(1/\epsilon^c)$ for all c?

The above represent specific incursions into the unknown square in Figure 2. Any other such incursion would also be interesting, and provide either for a memory lower bound, or a trade-off improving over Gradient Descent and Center of Mass in some regime.

Question 4 ($100 or Michelin Bib Gourmand Meal) Resolve the possibility or impossibility of some trade-off (T, M) polynomially inside the unknown square in Figure 2.

References

David S Atkinson and Pravin M Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69(1-3):1–43, 1995.

Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 109–115. ACM, 2002.

Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

Branko Grünbaum et al. Partitions of mass-distributions and of convex bodies by hyperplanes. *Pacific Journal of Mathematics*, 10(4):1257–1261, 1960.

Arkadii Semenovich Nemirovsky and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

Naum Z Shor. Convergence rate of the gradient descent method with dilatation of the space. *Cybernetics*, 6(2):102–108, 1970.

Blake Woodworth and Nathan Srebro. Lower bound for randomized first order convex optimization. *arXiv preprint arXiv:1709.03594*, 2017.

Appendix A. Analysis of Gradient Descent

Algorithm 1 Memory-Bounded Gradient Descent

Initialize: $x_0 = 0$, $x_{\text{best}} = 0$, $F_{\text{best}} = \infty$, $\mu_0 = \{x_0, x_{\text{best}}, F_{\text{best}}\}$
for $t = 0, \dots, T$ **do**
  Compute $F(x_t), \nabla F(x_t)$ } $\phi_t(\mu_t)$
  if $F(x_t) < F_{\text{best}}$ **then**
    $F_{\text{best}} = \text{Discretize}(F(x_t))$ {Discretize s.t. $F_{\text{best}} \le F(x_t)$}
    $x_{\text{best}} = x_t$
  $\tilde{x}_{t+1} = x_t - \eta_t \nabla F(x_t)$ } $\psi_t(\mu_t, \phi(\mu_t))$
  $x_{t+1} = \text{Discretize}(\tilde{x}_{t+1})$
  $\mu_{t+1} = \{x_{t+1}, x_{\text{best}}, F_{\text{best}}\}$
return x_{best}

Theorem 1 *For any L-Lipschitz and convex function F with $\|x^*\| \le B$, the gradient descent algorithm can find a point $\hat{x}$ with $F(\hat{x}) - F^* \le \epsilon$ using $O\left(d \log \frac{LB}{\epsilon}\right)$ bits of memory and $O\left(\frac{L^2 B^2}{\epsilon^2}\right)$ function and gradient evaluations.*

Proof

For now, assume that in each iteration the perturbation of the gradient descent iterates resulting from the discretization $\tilde{x}_t \mapsto x_t$ is bounded in L2 norm, i.e. $\|\tilde{x}_t - x_t\| \le D$. Then, following the standard gradient descent analysis,

$$\begin{aligned}
\|x_{t+1} - x^*\|^2 &= \|\tilde{x}_{t+1} - x^* + x_{t+1} - \tilde{x}_{t+1}\|^2 && (5)\\
&\le \|x_t - \eta_t \nabla F(x_t) - x^*\|^2 + D^2 + 2\left\langle \tilde{x}_{t+1} - x^*,\, x_{t+1} - \tilde{x}_{t+1} \right\rangle && (6)\\
&= \|x_t - x^*\|^2 + \eta_t^2 \|\nabla F(x_t)\|^2 - 2\eta_t \left\langle \nabla F(x_t),\, x_t - x^* \right\rangle + D^2 + 2DB && (7)\\
&\le \|x_t - x^*\|^2 + \eta_t^2 L^2 - 2\eta_t \left(F(x_t) - F^*\right) + D^2 + 2DB && (8)
\end{aligned}$$

Rearranging this expression, we conclude

$$F(x_t) - F^* \le \frac{1}{2\eta_t}\left(\|x_t - x^*\|^2 - \|x_{t+1} - x^*\|^2\right) + \frac{\eta_t L^2}{2} + \frac{D^2 + 2DB}{2\eta_t} \qquad (9)$$

Choosing a fixed stepsize $\eta_t = \eta = \frac{B}{L\sqrt{T}}$ and averaging the iterates, we conclude $\bar{x}_T = \frac{1}{T}\sum_{t=1}^{T} x_t$ achieves suboptimality

$$F(\bar{x}_T) - F^* \le \frac{1}{T}\sum_{t=1}^{T} F(x_t) - F^* \tag{10}$$

$$\le \frac{1}{T}\sum_{t=1}^{T} \frac{1}{2\eta}\left(\|x_t - x^*\|^2 - \|x_{t+1} - x^*\|^2\right) + \frac{\eta L^2}{2} + \frac{D^2 + 2DB}{2\eta} \tag{11}$$

$$= \frac{L}{2B\sqrt{T}}\left(\|x_1 - x^*\|^2 - \|x_{T+1} - x^*\|^2\right) + \frac{LB}{2\sqrt{T}} + \frac{(D^2 + 2DB)L\sqrt{T}}{2B} \tag{12}$$

$$\le \frac{LB}{\sqrt{T}} + \frac{(D^2 + 2DB)L\sqrt{T}}{2B} \tag{13}$$

Thus, $D \le \frac{B}{T}$ ensures

$$F(\bar{x}_T) - F^* \le \frac{LB}{\sqrt{T}} + \frac{(D^2 + 2DB)L\sqrt{T}}{2B} \tag{14}$$

$$\le \frac{LB}{\sqrt{T}} + \left(\frac{B}{2T^2} + \frac{2B}{2T}\right) L\sqrt{T} \tag{15}$$

$$\le \frac{3LB}{\sqrt{T}} \tag{16}$$

Since the averaged iterate achieves this suboptimality, the best iterate's suboptimality is at least this good. For $T \ge \frac{9L^2B^2}{\epsilon^2}$, this ensures that at least one of the iterates was ϵ-suboptimal. As long as F_{best} is discretized to accuracy ϵ, then x_{best} is at most 2ϵ-suboptimal. This discretization can be achieved using $\log \frac{2LB}{\epsilon}$ bits.

Discretizing the iterates up to accuracy $D = \frac{B}{T} = \frac{\epsilon^2}{9L^2B}$ can be achieved using the log of the $\frac{\epsilon^2}{9L^2B}$ L2 covering number of the radius-B ball, which is upper bounded by $d \log\left(1 + \frac{36L^2B^2}{\epsilon^2}\right)$ bits. The discretization of x_{best} is achieved using the same number of bits.

Therefore, the total number of bits of memory needed to implement gradient descent is at most

$$2 \cdot d \log\left(1 + \frac{36L^2B^2}{\epsilon^2}\right) + \log\frac{2LB}{\epsilon} = O\left(d \log\frac{LB}{\epsilon}\right) \tag{17}$$

■

Appendix B. Analysis of Center of Mass Algorithm

Lemma 2 (Grünbaum et al. (1960)) *For any convex set $K \subseteq \mathbb{R}^d$ with center of gravity c, and any halfspace $H = \{x : \langle a, x - c\rangle \ge 0\}$ passing through c,*

$$\frac{1}{e} \le \frac{\mathit{Vol}\,(K \cap H)}{\mathit{Vol}\,(K)} \le 1 - \frac{1}{e}$$

Algorithm 2 Memory-Bounded Center of Mass

Initialize: $K_0 = \{x : \|x\| \le B\}$, $\mu_0 = \varnothing$
for $t = 0, \dots, T$ **do**
  for $k = 1, \dots, t$ **do**
    $c_{k-1} = \int_{K_{k-1}} x dx / \int_{K_{k-1}} dx$
    $K_k = K_{k-1} \cap \left\{x : \left\langle \tilde{\nabla} F(c_{k-1}),\, x - c_{k-1} \right\rangle \le 0\right\}$
  $c_t = \int_{K_t} x dx / \int_{K_t} dx$ } $\phi_t(\mu_t)$
  $\tilde{F}(c_t), \tilde{\nabla} F(c_t) = \text{Discretize}\left(F(c_t), \nabla F(c_t)\right)$
  $\mu_{t+1} = \mu_t \cup \left\{\tilde{F}(c_t), \tilde{\nabla} F(c_t)\right\}$ } $\psi_t(\mu_t, \phi(\mu_t))$
return $\text{Discretize}\left(\arg\min_{c \in \{c_1, \dots, c_T\}} \tilde{F}(c)\right)$

Theorem 3 *For any L-Lipschitz and convex function F with $\|x^*\| \le B$, the center of mass algorithm can find a point $\hat{x}$ with $F(\hat{x}) - F^* \le \epsilon$ using $O\left(d^2 \log^2 \frac{LB}{\epsilon}\right)$ bits of memory and $O\left(d \log \frac{LB}{\epsilon}\right)$ function and gradient evaluations.*

Proof This proof is quite similar to existing analysis of the center of mass algorithm (Bubeck et al., 2015), we simply take care to count the number of required bits.

Consider the set $K^\alpha = \{(1-\alpha)x^* + \alpha x : x \in K_0\}$, which has volume $\text{Vol}\left(K^\alpha\right) = \alpha^d \text{Vol}\left(K_0\right)$. By convexity,

$$\begin{aligned} F((1-\alpha)x^* + \alpha x) &\le (1-\alpha)F^* + \alpha F(x) && (18)\\ &\le (1-\alpha)F^* + \alpha\left(F^* + \|\nabla F(x)\|\,\|x - x^*\|\right) && (19)\\ &\le F^* + 2\alpha LB && (20)\end{aligned}$$

By Grünbaum's Lemma, $\text{Vol}\left(K_T\right) \le \left(1 - \frac{1}{e}\right)\text{Vol}\left(K_{T-1}\right) \le \left(1 - \frac{1}{e}\right)^T \text{Vol}\left(K_0\right)$. Thus, when $T \ge 3d\log(1/\alpha)$

$$\text{Vol}\left(K_T\right) \le \left(1 - \frac{1}{\epsilon}\right)^T \text{Vol}\left(K_0\right) < \alpha^d \text{Vol}\left(K_0\right) = \text{Vol}\left(K^\alpha\right) \quad (21)$$

We conclude that there must be some iteration t in which $\exists y \in K^\alpha \cap (K_t \setminus K_{t+1})$. Thus, $y \in K^\alpha$ and $\left\langle \tilde{\nabla} F(c_t),\, y - c_t \right\rangle > 0$. We will now argue that the center of mass c_t has small error:

$$\begin{aligned} F(c_t) &\le F(y) + \langle \nabla F(c_t),\, c_t - y\rangle && (22)\\ &\le F^* + 2\alpha LB + \left\langle \nabla F(c_t) - \tilde{\nabla} F(c_t),\, c_t - y \right\rangle + \left\langle \tilde{\nabla} F(c_t),\, c_t - y \right\rangle && (23)\\ &\le F^* + 2\alpha LB + \left\|\nabla F(c_t) - \tilde{\nabla} F(c_t)\right\| \|c_t - y\| + 0 && (24)\\ &\le F^* + 2B\left(\alpha L + \left\|\nabla F(c_t) - \tilde{\nabla} F(c_t)\right\|\right) && (25)\end{aligned}$$

Therefore, if we choose $\alpha = \frac{\epsilon}{4LB}$ and discretize gradients with L2 error at most $\frac{\epsilon}{4B}$, this ensures that $F(c_t) - F^* \le \epsilon$.

The gradients of an L-Lipschitz function are contained in the Euclidean ball of radius L. Therefore, the gradients can be discretized with error $\frac{\epsilon}{4B}$ using the logarithm of the $\frac{\epsilon}{4B}$ L2 covering number of the Euclidean ball of radius L, which is upper bounded by $d\log\left(1+\frac{16LB}{\epsilon}\right)$ bits. There are $T = 3d\log(1/\alpha) = 3d\log\left(\frac{4LB}{\epsilon}\right)$ gradients in total, thus the total number of bits required to represent the gradients is

$$T \cdot d\log\left(1+\frac{16LB}{\epsilon}\right) = 3d^2\log\left(\frac{4LB}{\epsilon}\right)\log\left(1+\frac{16LB}{\epsilon}\right) \tag{26}$$

Since $\epsilon \le LB$, this is upper bounded by $3d^2\log^2\left(\frac{17LB}{\epsilon}\right)$ bits.

Once T iterations have been completed, we know that at least one of the centers must be an ϵ-approximate minimizer of the objective. Using the stored gradients, we can then recompute all centers and return a discretization of the best center. As long as $\left|\tilde{F}(c_t) - F(c_t)\right| \le \epsilon$ for all t, then the center chosen by the algorithm will be within ϵ of the best center. This discretization of the function values requires $T \cdot \log\left(\frac{2LB}{\epsilon}\right)$ bits.

As long as the discretization of the chosen center has L2 error at most ϵ/L, then the output will be a 3ϵ-approximate minimizer. The number of bits needed for this discretization is at most $d\log\left(1+\frac{4LB}{\epsilon}\right)$. Therefore, the total number of bits needed is at most

$$3d^2\log^2\left(\frac{17LB}{\epsilon}\right) + 3d\log\left(\frac{4LB}{\epsilon}\right)\log\left(\frac{2LB}{\epsilon}\right) + d\log\left(1+\frac{4LB}{\epsilon}\right) = O\left(d^2\log^2\left(\frac{LB}{\epsilon}\right)\right) \tag{27}$$

Rescaling $\epsilon' = \epsilon/3$ completes the proof. ■

Appendix C. Memory Lower Bound

Lemma 4 *The packing number of the Euclidean unit sphere in $\mathbb{R}^d$ with distance α is at least α^{-d}.*

Proof Let $\{x_1, \ldots, x_N\}$ be the largest possible packing of the unit sphere, with $N < \alpha^{-d}$. Consider the set of points that are within α of one of the points in the packing:

$$\begin{aligned}
\text{Vol}\left(x : \exists i \, \|x - x_i\| \le \alpha\right) &= \text{Vol}\left(\bigcup_i B(x_i, \alpha)\right) && (28)\\
&\le \sum_i \text{Vol}\left(B(x_i, \alpha)\right) && (29)\\
&= N\alpha^d \text{Vol}\left(B(0,1)\right) && (30)\\
&< \text{Vol}\left(B(0,1)\right) && (31)
\end{aligned}$$

Therefore, there exists a point $y \in B(0,1)$ such that $\|y - x_i\| > \alpha$ for all i. The existence of such a point contradicts the assumption that $\{x_1, \ldots, x_N\}$ is the largest possible packing. We conclude that the packing number is at least α^{-d}. ■

Theorem 5 *For any $L, B > 0$ and any $\epsilon \leq \frac{LB}{2}$, any optimization algorithm that is guaranteed to return an ϵ-suboptimal point for any convex, L-Lipschitz function with $\|x^*\| \leq B$ must use at least $d \log \frac{LB}{2\epsilon}$ bits of memory.*

Proof To begin, by Lemma 4 there exists a packing $\{x_1, \ldots, x_N\}$ of the ball $\{x : \|x\| \leq B\}$ of size at least $N \geq \left(\frac{LB}{2\epsilon}\right)^d$ such that $\|x_i - x_j\| > \frac{2\epsilon}{L}$ for all $i \neq j \in [N]$. We will associate a function with each point in the packing, let

$$f_i(x) = L \|x - x_i\| \tag{32}$$

These functions are convex and L-Lipschitz, and their optimizers x_i have norm less than B.

Note that any point x which is an approximate minimizer of some f_i must have high function value on all other functions f_j. Suppose $f_i(x) \leq \epsilon$, then $\|x - x_i\| \leq \frac{\epsilon}{L}$. Consequently, for all $j \neq i$, $\|x - x_j\| = \|x - x_i + x_i - x_j\| \geq \|x_i - x_j\| - \|x - x_i\| > \frac{\epsilon}{L}$, thus $f_j(x) = L \|x - x_j\| > \epsilon$.

Consider using a memory-bounded optimization algorithm to optimize one of these functions f_i. After the algorithm has made all of its first-order oracle accesses, the output function ζ must map from the final memory state μ_T to a solution $\hat{x}$. Suppose the final memory state μ_T uses $M < \log N$ bits, then there are at most $2^M < N$ outputs that the algorithm might give. However, as we just argued, there exist N functions such that returning an accurate solution for any one of them requires returning an inaccurate solution for all the others. Consequently, any algorithm which can output fewer than N different outputs will fail to optimize at least one of the functions $f_1, \ldots, f_N$. ■